M55M1 Smart Motion Detection

NuMicro® 32 位元系列微控制器範例代碼介紹

文件信息

应用简述	本范例代码展示如何在M55M1上结合使用低功耗模式下的动作侦测功能。根据对象侦测结果，系统将动态地启用或屏蔽动作侦测区块，以达到最佳效能和节能效果。
BSP 版本	M55M1_Series_BSP_CMSIS_V3.00.001
开发平台	NuMaker-M55M1(BETA) V1.1

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1. 概述

在本范例代码中，展示了如何在 M55M1 微控制器上结合使用低功耗模式（Low Power Mode）和动作侦测功能（Motion Detection Feature）。此范例的核心目标是根据人的对象侦测（Object Detection）结果，智慧地控制动作侦测区块的开启或屏蔽，从而达到节能效果。

具体来说，系统首先利用对象侦测技术来分析场景中的物体。一旦侦测到特定物体，系统会根据预先设定的条件决定是否启用或屏蔽对应的动作侦测区块。例如，如果人进入某个区域，动作侦测功能将被启动；反之，如果没有侦测到人，系统会屏蔽相关区块以节省电能。

这种方法能够有效地优化系统资源的使用，特别适合于需要长时间运行且受限于电池容量的应用场景，如智能监控、家居安防等。

1.1 原理

本范例展示了如何利用 M55M1 微控制器和低功耗模式来实现高效的动作侦测功能。通过结合对象侦测技术，系统能够智能地控制动作侦测区块的开启或屏蔽，从而达到最佳效能和节能效果。这种方法在智慧设备应用中具有广泛的潜力，值得进一步研究和推广。

整体流程如所示圖 1-1。

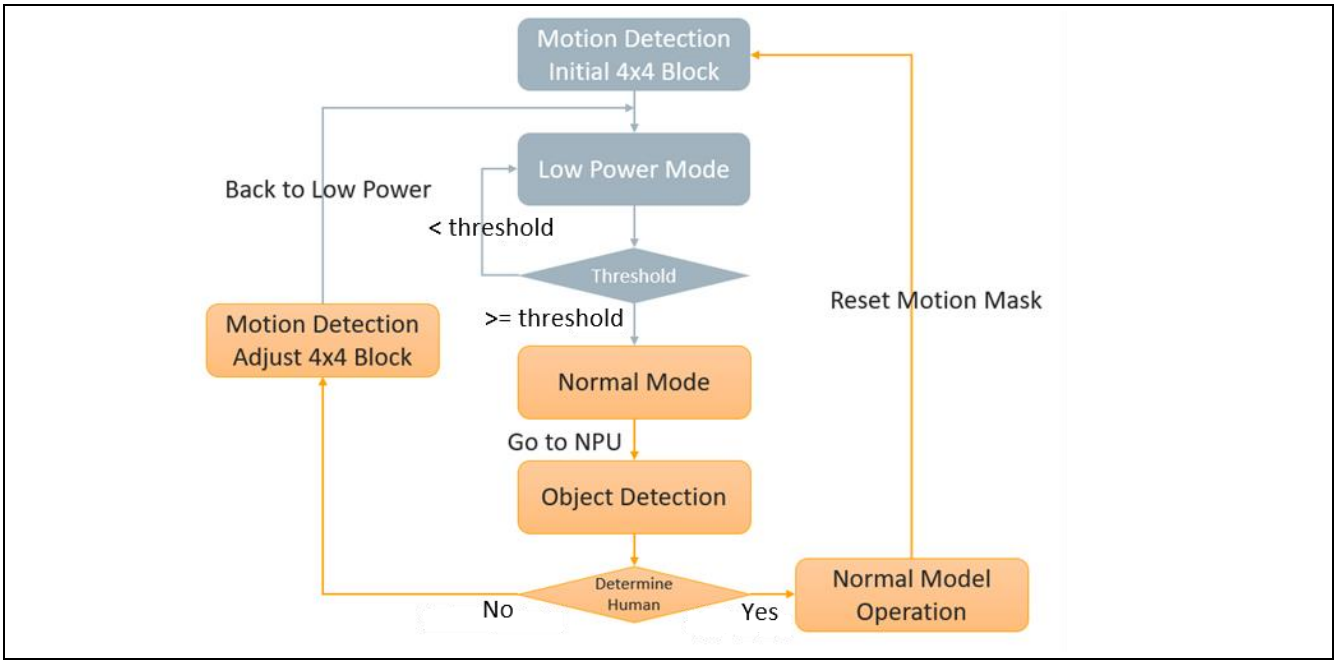

圖 1-1 Smart Motion Detection 流程图

1.2 流程详述

这个系统基于省电模式进行动作侦测，整合了相机撷取控制器（CCAP）与芯片的功能。在系统启动时，芯片默认处于省电模式，只有在定时器触发后，才开始进行动作侦测。首先，CCAP 撷取一张 640x480 的 Y 格式影像，并缩小至 320x240 的影像。接着，透过最大池化处理 (Max Pooling) 将影像进一步缩小为 160x120。然后，系统从该影像中提取 20x15 的子影像，共 16 个窗口。通过影像比对算法计算绝对差值总和 (SAD)，来生成最终的动作侦测结果。系统随后检查 CCAP 中断状态，若中断条件成立，则唤醒 CPU 并触发中断；若不成立，则系统返回省电模式，以确保能源效率。

当检测到的运动能量小于设定的阈值时，系统会保持在低功耗模式（Low Power Mode），以节省能源。然而，当检测到的运动能量大于阈值时，系统会 Wake up 进入正常模式（Normal Mode）并启动 NPU 做目标检测。

在正常模式下，神经网络处理单元（NPU）会进行目标检测。这里使用的是 YOLO 模型（You Only Look Once），这是一种快速而准确的目标检测算法。训练好的 YOLO 模型可以判断图像中的对像是人还是其他物体。

当系统判断图像中存在人时，会进入用户操作模式，允许用户进行相关操作。另一方面，当系统判断图像中不存在人时，会进行 4x4 区块的调整，将误报（False Alarm）的区块关闭，并回到 Motion Detection 的初始 4x4 区块阶段，重新开始检测过程，一直到系统判断图像中存在人时，被关闭的 4x4 区块会被重新打开，Motion Detection 会回到初始的状态。

这样的设计可以有效地平衡功耗和检测精度，确保系统在不同环境下的稳定运行。

- 动作侦测初始设置（Motion Detection Initial 4x4 Block）

  系统首先启动动作侦测功能，并将其设置为初始的 4x4 区块大小。这个设置允许系统在一个小范围内进行初步侦测，以便节省电力。

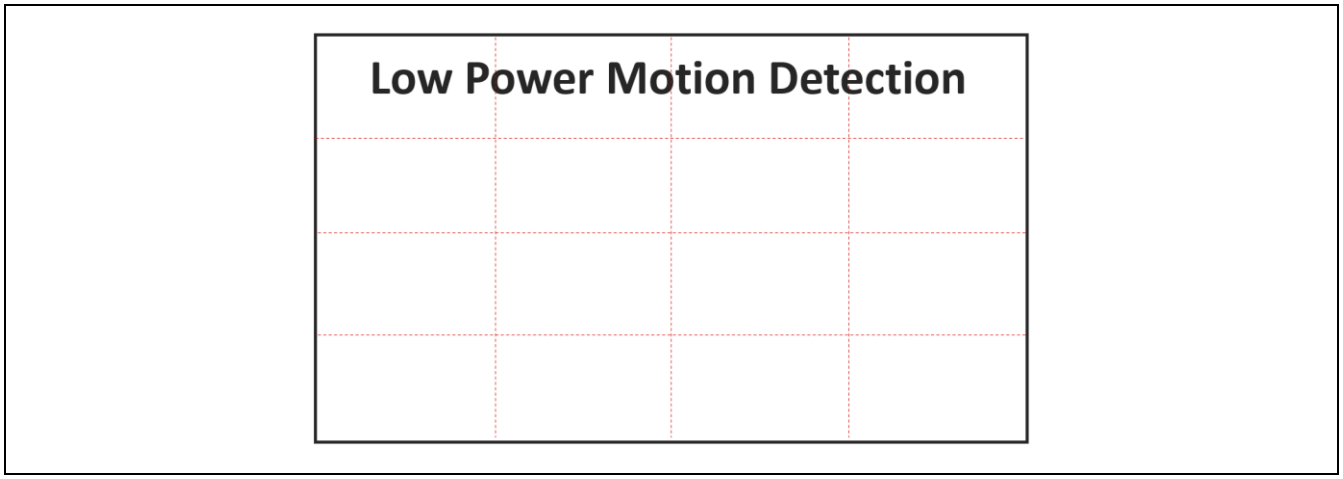

圖 1-2 Motion Detection Initial 4x4 Block

- 低功耗模式（Low Power Mode）

  在初始设置完成后，系统进入低功耗模式。在这种模式下，动作侦测功能以最低的能耗运行，目的是延长电池寿命和减少能源消耗。

- 门坎值判断（Threshold）

  系统对侦测到的动作信号进行评估，并与预设的门坎值进行比较：

  - 如果侦测到的动作信号强度小于门坎值（< Th），系统将继续运行在低功耗模式。
  - 如果侦测到的动作信号强度大于门坎值（> Th），系统将切换到正常模式进行更高级别的侦测，并调整 4x4 区块的设置，以适应新的环境条件。

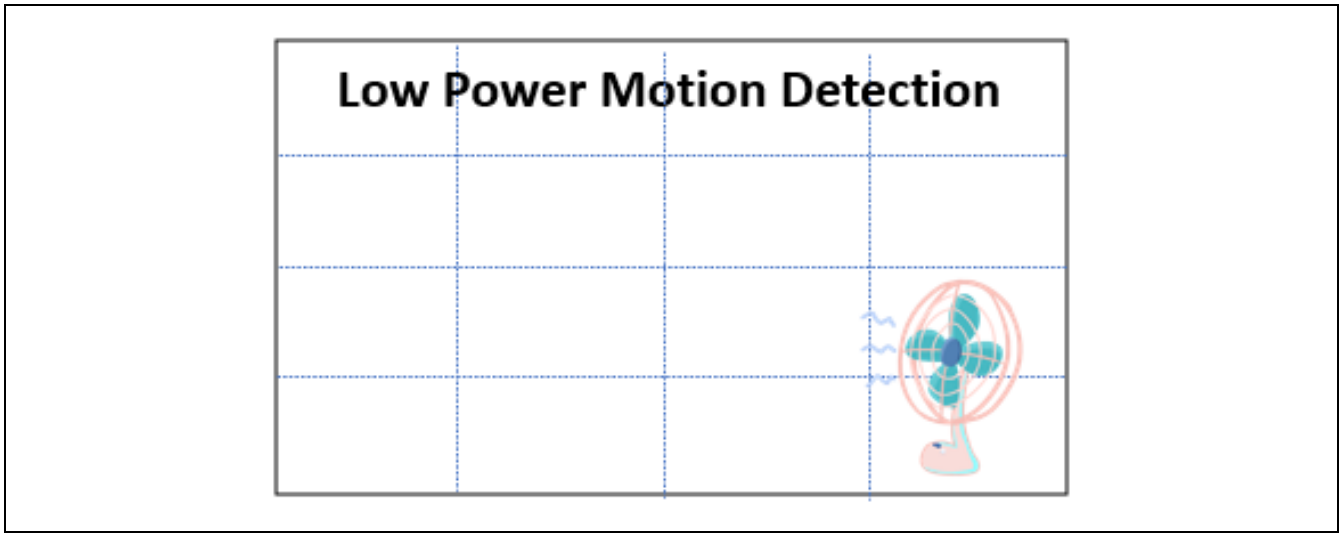

圖 1-3 当系统有侦测到 Motion 的情况

- 正常模式（Normal Mode）

  当信号强度超过门坎值时，系统进入正常模式。在此模式下，动作侦测功能全面启动，能够更精确地侦测到场景中的变化。

- 对象侦测（Object Detection）

  在正常模式下，侦测到的数据将被传送到 NPU（神经处理单元）进行对象侦测。NPU 利用深度学习算法来分析数据，并确定侦测到的物体是什么。

- 判断是否为人类（Determine Human）

  根据对象侦测的结果，系统将判断侦测到的物体是否为人类：

  如果侦测结果显示为人类（is human），系统将进入正常模式操作（Normal Mode Operation），继续执行当前任务并保持系统设置。

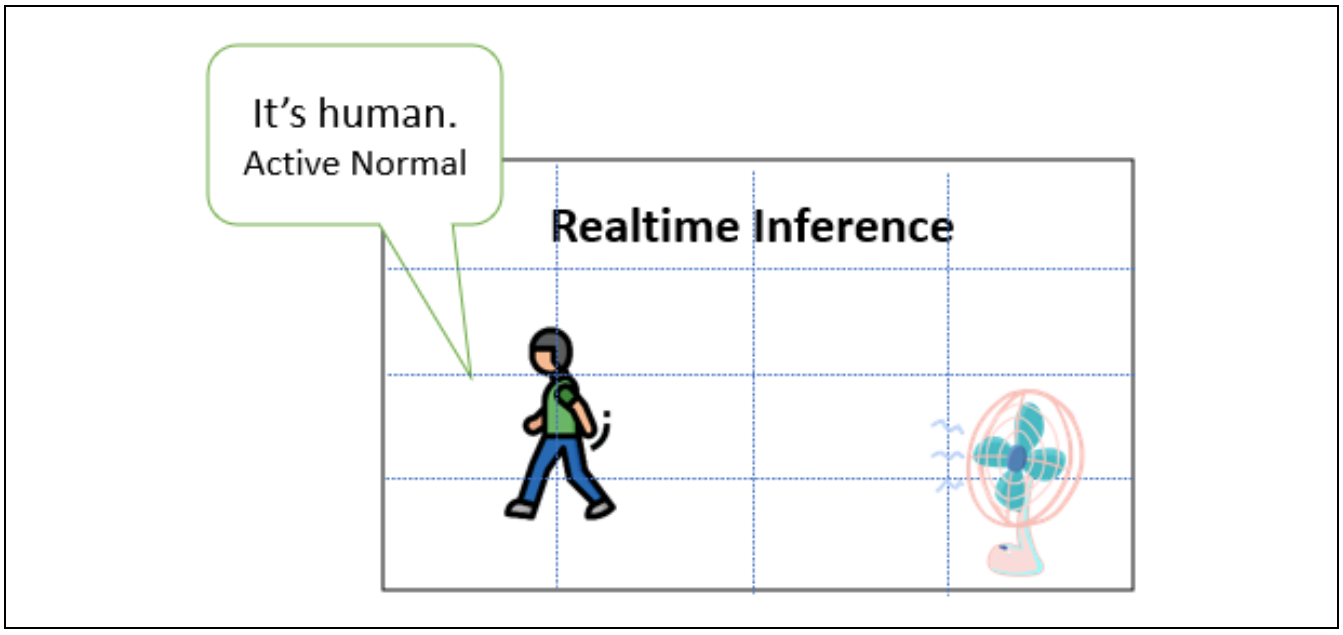

图 1-4 侦测到的物体是人

如果检测结果显示不是人类（is not human），系统将返回调整 4x4 块设置，并重新进入低功耗模式以继续下一轮检测。

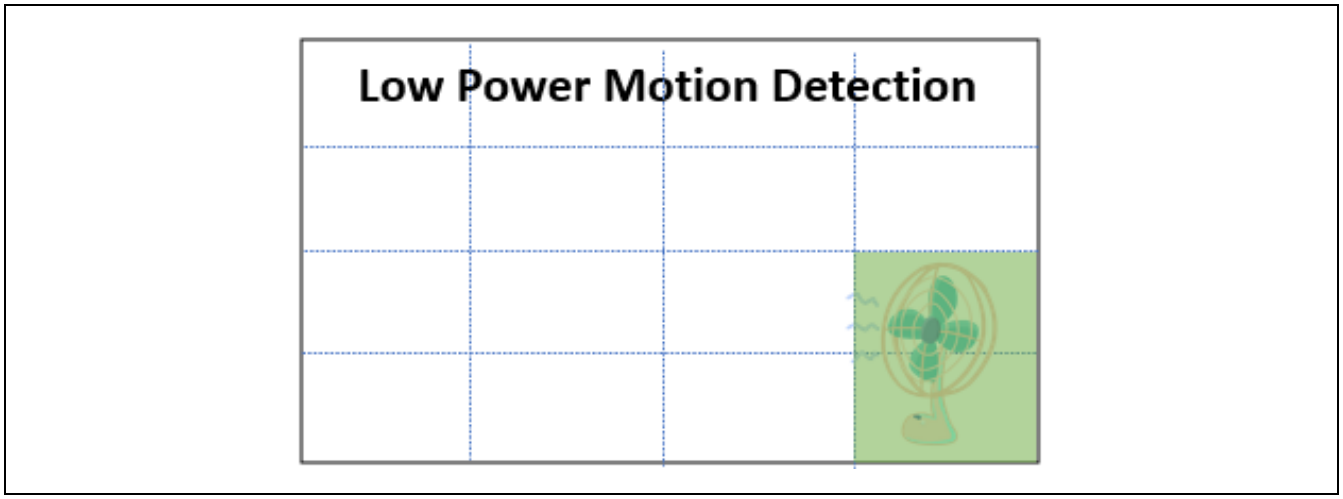

图 1-5 系统屏蔽掉不是人的区块

1.3 使用 Pre-Trained Model

在进行目标检测时，YOLO（You Only Look Once）模型因其高效的推理速度和良好的精度而广受欢迎。YOLO-Fastest-1.1 是 YOLO 家族中的一个轻量级变体，专为嵌入式设备和计算资源有限的环境设计。本文将详细描述使用 YOLO-Fastest-1.1 模型及 COCO 数据集进行训练和应用的过程和结果。

1.3.1 YOLO-Fastest-1.1 模型简介

YOLO-Fastest-1.1 是基于 YOLOv3 的改进版本，旨在降低计算资源的需求，同时保持一定的检测精度。这个模型的主要特点是参数量小、推理速度快，适合在移动设备和嵌入式系统上进行实时目标检测。YOLO-Fastest-1.1 在设计上采用了更轻量的网络架构，如使用 shufflenetV2 作为骨干网，并引入了 YOLOv5 的锚点匹配机制和损失函数，使得模型能在保持速度的同时减少参数量。

1.3.2 COCO 数据集简介

COCO（Common Objects in Context）数据集是目标检测和图像分割领域中最常用的数据集之一。它包含了 80 个类别的物体，涵盖了日常生活中常见的各种场景和物品。每张图像都附有丰富的标注信息，包括物体的边界框、语义分割掩码和关键点等。这使得 COCO 数据集成为训练和评估目标检测模型的理想选择。

COCO 数据集包含以下内容：

- 训练集：包含约 118,000 张图像及其标注。
- 验证集：包含约 5,000 张图像及其标注。
- 测试集：包含约 40,000 张图像，但没有标注。

1.3.3 Label 的种类

编号	类别名称	编号	类别名称	编号	类别名称	编号	类别名称
1	person	21	elephant	41	wine glass	61	dining table
2	bicycle	22	bear	42	cup	62	toilet
3	car	23	zebra	43	fork	63	tv
4	motorcycle	24	giraffe	44	knife	64	laptop
5	airplane	25	backpack	45	spoon	65	mouse
6	bus	26	umbrella	46	bowl	66	remote
7	train	27	handbag	47	banana	67	keyboard
8	truck	28	tie	48	apple	68	cell phone
9	boat	29	suitcase	49	sandwich	69	microwave
10	traffic light	30	frisbee	50	orange	70	oven
11	fire hydrant	31	skis	51	broccoli	71	toaster
12	stop sign	32	snowboard	52	carrot	72	sink
13	parking meter	33	sports ball	53	hot dog	73	refrigerator
14	bench	34	kite	54	pizza	74	book
15	bird	35	baseball bat	55	donut	75	clock
16	cat	36	baseball glove	56	cake	76	vase
17	dog	37	skateboard	57	chair	77	scissors
18	horse	38	surfboard	58	couch	78	teddy bear
19	sheep	39	tennis racket	59	potted plant	79	hair drier
20	cow	40	bottle	60	bed	80	toothbrush

1.3.4 Person 的类别

"person" 类别是最常见和最重要的类别之一。该数据集旨在提供丰富的标注数据以支持目标检测、语义分割和图像标注等任务。COCO 数据集涵盖了日常生活中的各种场景，并且每张图像都附有详细的标注信息，其中包括物体的边界框、语义分割掩码和关键点等。

边界框（Bounding Box）：

边界框标注了每个人在图像中的位置和大小。这些框是以四个坐标值（x, y, w, h）来表示的，其中 (x, y) 是边界框左上角的坐标，w 和 h 分别是边界框的宽度和高度。

例如，一个 “person” 类别的标注档可能包含以下内容：

```
0 0.312 0.456 0.234 0.567
```

其中，0 是类别索引，0.312 和 0.456 是边界框中心的归一化坐标，0.234 和 0.567 是边界框的宽度和高度。

1.4 执行结果

本范例代码执行编译和烧录的结果如图 1-6 所示。

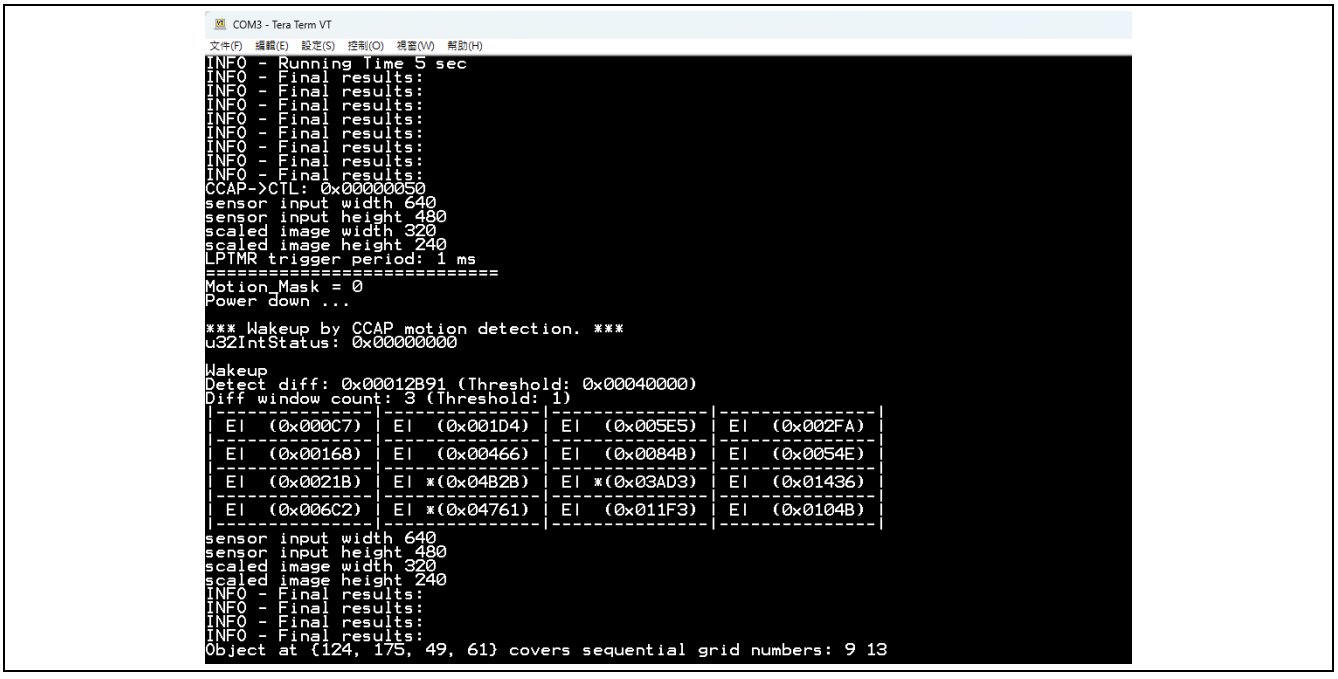

图 1-6 执行编译和烧录结果

执行结果会由 PC UART 打印出 Object Detection 的结果，以及侦测到 Motion 区块等信息。

以下为 Object Detection 打印结果，结束后会回到 Power Down 的模式。

```
INFO - BoardInit: complete
INFO - Ethos-U device initialised
INFO - Ethos-U version info:
INFO -  Arch:       v1.1.0
INFO -  Driver:     v0.16.0
INFO -  MACs/cc:    256
INFO -  Cmd stream: v0
INFO - Target system: M55M1
INFO - main task running
INFO - Set tesnor arena cache policy to WTRA
u32RegionCnt: 3, (MPU_REGIONS_MAX - i32RegionIdx): 7
INFO - Added ethos-u support to op resolver
INFO - Creating allocator using tensor arena at 0x82000000
INFO - Allocating tensors
INFO - Model INPUT tensors:
INFO -  tensor type is INT8
INFO -  tensor occupies 307200 bytes with dimensions
INFO -          0:   1
INFO -          1: 320
INFO -          2: 320
INFO -          3:   3
INFO - Quant dimension: 0
INFO - Scale[0] = 0.003922
INFO - ZeroPoint[0] = -128
INFO - Model OUTPUT tensors:
INFO -  tensor type is INT8
INFO -  tensor occupies 102000 bytes with dimensions
INFO -          0:   1
INFO -          1:  20
INFO -          2:  20
INFO -          3: 255
INFO - Quant dimension: 0
INFO - Scale[0] = 0.190095
INFO - ZeroPoint[0] = 77
INFO -  tensor type is INT8
INFO -  tensor occupies 25500 bytes with dimensions
INFO -          0:   1
INFO -          1:  10
INFO -          2:  10
INFO -          3: 255
INFO - Quant dimension: 0
INFO - Scale[0] = 0.220456
INFO - ZeroPoint[0] = 67
INFO - Activation buffer (a.k.a tensor arena) size used: 1229496
INFO - Number of operators: 3
INFO -  Operator 0: ethos-u
INFO -  Operator 1: RESIZE_NEAREST_NEIGHBOR
INFO -  Operator 2: ethos-u
INFO - init perfcycles 900001188
Sensor Chip_Version_H = 0x09 (0x09) Chip_Version_L = 0x55 (0x55)
```

```
sensor input width 640
sensor input height 480
scaled image width 320
scaled image height 240
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Running Time 5 sec
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Final results:
CCAP->CTL: 0x00000050
sensor input width 640
sensor input height 480
scaled image width 320
scaled image height 240
LPTMR trigger period: 1 ms
============================
Motion_Mask = 0
Power down ...
```

如果有 Motion，会打印 Motion 的信息如下:

```
*** Wakeup by CCAP motion detection. ***
u32IntStatus: 0x00000000

Wakeup
Detect diff: 0x0000E3DB (Threshold: 0x00040000)
Diff window count: 3 (Threshold: 1)
|---------------|---------------|---------------|---------------|
| E|  (0x002CA) | E|  (0x00262) | E|  (0x00203) | E|  (0x001C8) |
|---------------|---------------|---------------|---------------|
| E|  (0x00203) | E|  (0x003F2) | E|  (0x00248) | E|  (0x00478) |
|---------------|---------------|---------------|---------------|
| E|  (0x009D9) | E|  (0x002FB) | E|  (0x005A2) | E|  (0x00AB7) |
|---------------|---------------|---------------|---------------|
| E| *(0x02A66) | E| *(0x0428C) | E|  (0x018A1) | E| *(0x02B6F) |
|---------------|---------------|---------------|---------------|
sensor input width 640
sensor input height 480
scaled image width 320
```

```
scaled image height 240
INFO - Final results:
Object at {68, 169, 61, 70} covers sequential grid numbers: 8 12
INFO - Final results:
```

如果有侦测到对象，就会如上信息，打印出 Object 的位置，与 Label 的号码:

Object at {68, 169, 61, 70} covers sequential grid numbers: 8 12

因为程序代码设定的 Threshold 是 0x2000（可以参考以下第 2 章代码介绍的运动检测设定）被 Activate 的 window，会被标注 * 符号，如下打印 Motion 的信息所示。

```
|---------------|---------------|---------------|---------------|
| E|  (0x002CA) | E|  (0x00262) | E|  (0x00203) | E|  (0x001C8) |
|---------------|---------------|---------------|---------------|
| E|  (0x00203) | E|  (0x003F2) | E|  (0x00248) | E|  (0x00478) |
|---------------|---------------|---------------|---------------|
| E|  (0x009D9) | E|  (0x002FB) | E|  (0x005A2) | E|  (0x00AB7) |
|---------------|---------------|---------------|---------------|
| E| *(0x02A66) | E| *(0x0428C) | E|  (0x018A1) | E| *(0x02B6F) |
|---------------|---------------|---------------|---------------|
```

关于 grid numbers 的编号，如下所示:

```
|---------------|---------------|---------------|---------------|
|       0       |       1       |       2       |       3       |
|---------------|---------------|---------------|---------------|
|       4       |       5       |       6       |       7       |
|---------------|---------------|---------------|---------------|
|       8       |       9       |      10       |      11       |
|---------------|---------------|---------------|---------------|
|      12       |      13       |      14       |      15       |
|---------------|---------------|---------------|---------------|
```

2. 代码介绍

初始化 motion detection 的 4x4 Block，代码位于 *main.cpp*。

```
//Power down and CCAP wakeup procedure
/* Config CCAP motion detection */
if (PacketMotionDetection(&g_sSensorHM1055_VGA_YUV422) != 0)
{
    printf("Test motion detection failed !\n");

}
ImageSensor_Config(eIMAGE_FMT_RGB565, CCAP_MD_WIDTH, CCAP_MD_HEIGHT);
/* Init CCAP clock and Sensor clock */
CLK_SetModuleClock(CCAP0_MODULE, CLK_CCAPSEL_CCAP0SEL_HIRC, 0);
/* Config LPTMR0 to trigger CCAP motion detection per 100 ms */
ConfigLPTMR(LPTMR0, 1);
CCAP_MD_SET_TRIG_SRC(CCAP_MD_TRIG_LPTMR0, TRUE);

printf("============================\n");
printf("Motion_Mask = %x \n", Motion_Mask);
CCAP_MD_ENABLE_WINDOW(0xFFFF);
CCAP_MD_DISABLE_WINDOW(Motion_Mask);

PMC_SetPowerDownMode(PMC_NPD0, PMC_PLCTL_PLSEL_PL1);
EnterPowerDown();

CLK_SetModuleClock(CCAP0_MODULE, CLK_CCAPSEL_CCAP0SEL_HXT, 0);
SPIM_HYPER_EnableDLL(HYPERRAM_SPIM_PORT);
ImageSensor_Config(eIMAGE_FMT_RGB565, frameBuffer.w, frameBuffer.h); //eIMAGE_FMT_RGB565

u64PowerDownCycle = (uint64_t)pmu_get_systick_Count() + (uint64_t)(SystemCoreClock *
POWERDOWN_SEC);
```

以下是详细讲解实际使用 Smart Motion Detection 的方法。

程序代码实现了嵌入式系统上的实时对象侦测和运动检测功能。通过对模型初始化、图像处理、运动检测设置、结果显示和省电模式管理的详细解说，展示了如何在资源有限的环境中有效地执行复杂的图像处理任务。这些技术的应用有助于实现低功耗、高效能的嵌入式视觉系统。

模型初始化：

```
/* 模型对象创建和初始化 */
arm::app::YoloFastestModel model;

if (!model.Init(arm::app::tensorArena,
                sizeof(arm::app::tensorArena),
                arm::app::yolofastest::GetModelPointer(),
                arm::app::yolofastest::GetModelLen())) {
    printf_err("Failed to initialise model\n");
    return;
```

```
}

TfLiteTensor *inputTensor = model.GetInputTensor(0);
TfLiteTensor *outputTensor = model.GetOutputTensor(0);

if (!inputTensor->dims) {
    printf_err("Invalid input tensor dims\n");
    return;
} else if (inputTensor->dims->size < 3) {
    printf_err("Input tensor dimension should be >= 3\n");
    return;
}

TfLiteIntArray *inputShape = model.GetInputShape(0);

const int inputImgCols = inputShape->data[arm::app::YoloFastestModel::ms_inputColsIdx];
const int inputImgRows = inputShape->data[arm::app::YoloFastestModel::ms_inputRowsIdx];
}
```

在这一部分，程序代码首先创建并初始化 YOLO 模型对象。model.Init 函数使用张量缓冲区（tensorArena）来初始化模型，并通过调用 GetModelPointer 和 GetModelLen 来获取模型的指针和长度。接着，程序代码获取模型的输入张量（inputTensor）和输出张量（outputTensor），并检查输入张量的维度是否有效。如果输入张量的维度小于三，则输出错误信息并返回。

图像处理：

```
static void DrawImageDetectionBoxes(
    const std::vector<arm::app::object_detection::DetectionResult> &results,
    image_t *drawImg,
    std::vector<std::string> &labels) {
    for (const auto &result : results) {
        imlib_draw_rectangle(drawImg, result.m_x0, result.m_y0, result.m_w, result.m_h,
COLOR_B5_MAX, 1, false);
        imlib_draw_string(drawImg, result.m_x0, result.m_y0 - 16,
labels[result.m_cls].c_str(), COLOR_B5_MAX, 2, 0, 0, false,
                          false, false, false, 0, false, false);
    }
}
```

此函数负责在影像上绘制检测到的物体边界框和标签。函数接收检测结果、影像和标签列表，并在影像上绘制每个检测到的物体的边界框（使用 imlib_draw_rectangle 函数）和标签（使用 imlib_draw_string 函数）。

运动检测设定：

```
int32_t PacketMotionDetection(S_SENSOR_INFO *psSensorInfo) {
    int32_t i32RetCode = 0;
    int32_t i;

    /* 配置 CCAPEN(CCAP_CTL[0]) 以禁用捕获界面 */
    printf("CCAP->CTL: 0x%08X\n", CCAP->CTL);
    CCAP_Close();
    CCAP->CTL |= CCAP_CTL_VPRST_Msk;
    CLK_SysTickDelay(1000);
    CCAP->CTL &= ~CCAP_CTL_VPRST_Msk;

    /* 配置运动检测功能的控制寄存器 */
    CCAP_MD_SET_TOTAL_THRESHOLD(0x40000);

    /* 设置窗口运动检测阈值，最大值为 0x12AD4 */
    for (i = 0; i < 16; i++)
        CCAP_MD_SET_WIN_THRESHOLD(i, 0x2000);

    /* 设置溢出窗口计数阈值，最大值为 15。 */
    CCAP_MD_SET_OVERFLOW_WIN_THRESHOLD(1); // 8

    /* 启用 CCAP 运动检测中断 */
    CCAP_EnableInt(CCAP_INTSTS_MDINTF_MODE2_Msk);
    /* 启用外部捕获中断 */
    NVIC_EnableIRQ(CCAP_IRQn);

    return i32RetCode;
}
```

此函数负责配置 CCAP 的运动检测功能。首先，禁用捕获界面，然后设置全局运动检测阈值和每个窗口的运动检测阈值。接着，设置溢出窗口计数阈值，并启用 CCAP 运动检测中断和外部捕获中断。

结果显示：

```
static bool PresentInferenceResult(const
std::vector<arm::app::object_detection::DetectionResult> &results,
                                   std::vector<std::string> &labels) {
    info("Final results:\n");
    for (const auto &result : results) {
        // 如果不是人
        if (strcmp(labels[result.m_cls].c_str(), "person") != 0) {
            int left = result.m_x0;
            int top = result.m_y0;
            int right = result.m_x0 + result.m_w;
            int bottom = result.m_y0 + result.m_h;

            // 确保边界坐标不超出影像尺寸
            left = std::max(0, std::min(left, 319));
            top = std::max(0, std::min(top, 239));
            right = std::max(0, std::min(right, 319));
```

```
            bottom = std::max(0, std::min(bottom, 239));

            std::set<int> coveredGrids;
            for (int y = top; y <= bottom; y += 60) { // 在矩形高度范围内迭代
                for (int x = left; x <= right; x += 80) { // 在矩形宽度范围内迭代
                    auto grid_pos = getGridPosition(x, y);
                    coveredGrids.insert(getSequentialNumber(grid_pos.first,
grid_pos.second));
                }
            }

            // 输出所有被矩形覆盖的网格序列号
            printf("Object at {%d, %d, %d, %d} covers sequential grid numbers: ", left,
top, right - left, bottom - top);
            for (int num : coveredGrids) {
                printf("%d ", num);
                switch(num) {
                    case 0: Motion_Mask = Motion_Mask | 0x0001; break;
                    case 1: Motion_Mask = Motion_Mask | 0x0002; break;
                    case 2: Motion_Mask = Motion_Mask | 0x0004; break;
                    case 3: Motion_Mask = Motion_Mask | 0x0008; break;
                    case 4: Motion_Mask = Motion_Mask | 0x0010; break;
                    case 5: Motion_Mask = Motion_Mask | 0x0020; break;
                    case 6: Motion_Mask = Motion_Mask | 0x0040; break;
                    case 7: Motion_Mask = Motion_Mask | 0x0080; break;
                    case 8: Motion_Mask = Motion_Mask | 0x0100; break;
                    case 9: Motion_Mask = Motion_Mask | 0x0200; break;
                    case 10: Motion_Mask = Motion_Mask | 0x0400; break;
                    case 11: Motion_Mask = Motion_Mask | 0x0800; break;
                    case 12: Motion_Mask = Motion_Mask | 0x1000; break;
                    case 13: Motion_Mask = Motion_Mask | 0x2000; break;
                    case 14: Motion_Mask = Motion_Mask | 0x4000; break;
                    case 15: Motion_Mask = Motion_Mask | 0x8000; break;
                }
            }
            printf("\n");
        } else {
            // 如果是人
            CCAP_MD_ENABLE_WINDOW(0xFFFF);
            Motion_Mask = 0x0;
        }
    }
    return true;
}
```

此函数用于处理推论结果，并根据结果更新 Motion_Mask。如果检测到的物体不是人，则计算其覆盖的网格并更新 Motion_Mask；如果检测到的是人，则重置 Motion_Mask。

省电模式管理：

```
static void EnterPowerDown(void) {
    printf("Power down ...\n");
    UART_WAIT_TX_EMPTY(DEBUG_PORT);

    /* 进入省电模式时切换 SCLK 到 HIRC */
```

```
    CLK_SetBusClock(CLK_SCLKSEL_SCLKSEL_HIRC, CLK_APLLCTL_APLLSRC_HIRC, 0);
    /* 进入省电模式 */
    PMC_PowerDown();

    /* 设置处理器使用睡眠作为其低功耗模式 */
    SCB->SCR &= ~SCB_SCR_SLEEPDEEP_Msk;

    /* 启用 PLL0 180MHz 时钟从 HIRC 并切换 SCLK 时钟源到 PLL0 */
    CLK_SetBusClock(CLK_SCLKSEL_SCLKSEL_APLL0, CLK_APLLCTL_APLLSRC_HIRC, FREQ_180MHZ);
    printf("\nWakeup\n");

#if 1   /* 检查帧差异 */
    int32_t i;
    printf("Detect diff: 0x%08X (Threshold: 0x%08X)\n", CCAP->MDTSAD, CCAP->MDTTH);
    printf("Diff window count: %d (Threshold: %d)\n", CCAP_MD_GET_OVERFLOW_WIN_CNT(),
CCAP->MDWOCTH);

    for (i = 0; i < CCAP_MD_WINDOW_CNT; i++) {
        if (i % 4 == 0) {
            printf("|---------------|---------------|---------------|---------------|\n");
            printf("|");
        }

        if (CCAP_MD_GET_WIN_SAD(i) > CCAP->MDWTH[i])
            printf(" %s| *(0x%05X) |", (CCAP->MDCTL & (1 << i)) ? "E" : "D",
CCAP_MD_GET_WIN_SAD(i));
        else
            printf(" %s|  (0x%05X) |", (CCAP->MDCTL & (1 << i)) ? "E" : "D",
CCAP_MD_GET_WIN_SAD(i));

        if ((i + 1) % 4 == 0)
            printf("\n");
    }

    printf("|---------------|---------------|---------------|---------------|\n");
#endif
}
```

此函数管理嵌入式系统的省电模式。它首先切换系统时钟到高频内部振荡器（HIRC），然后进入省电模式。唤醒后，系统重新配置时钟，并检查帧差异以确保运动检测功能正常。

3. 软件与硬件需求

3.1 软件需求

- BSP 版本
  - M55M1_Series_BSP_CMSIS_V3.00.001
- IDE 版本
  - Keil uVersion 5.38

3.2 硬件需求

- 电路组件
  - NuMaker-M55M1(BETA) V1.1
- 线路示意图
  - 将 UART6 TX (PH.4) pin 脚连接到 PC UART RX，以显示范例代码的执行结果。

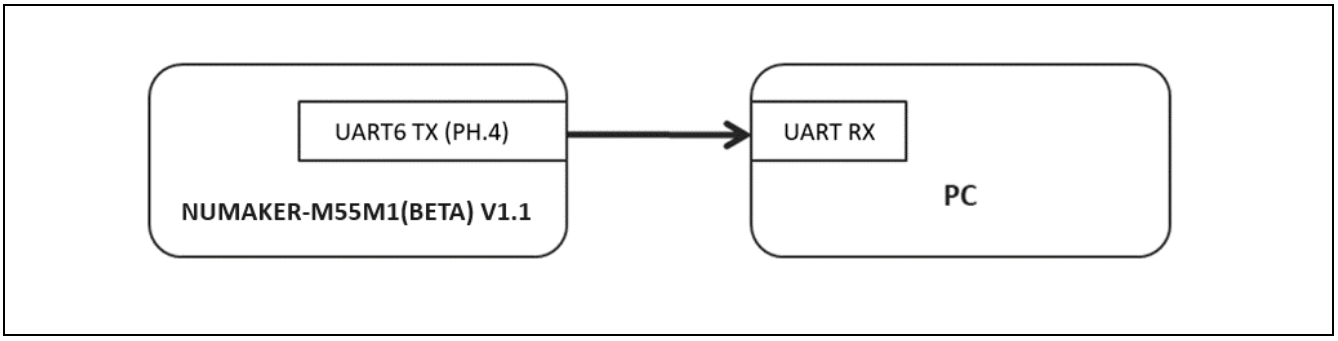

图 3-1 线路示意图

  - M55M1 使用 I2C 来传输控制命令至 CMOS 传感器，使用 CCAP_SCLK 作为 SCLK 供应频率给 CMOS 传感器，CMOS 传感器会根据设定回传 Pixel Clock (PIXCLK)、HSYNC、VSYNC 和 D0~D7 数据数据。

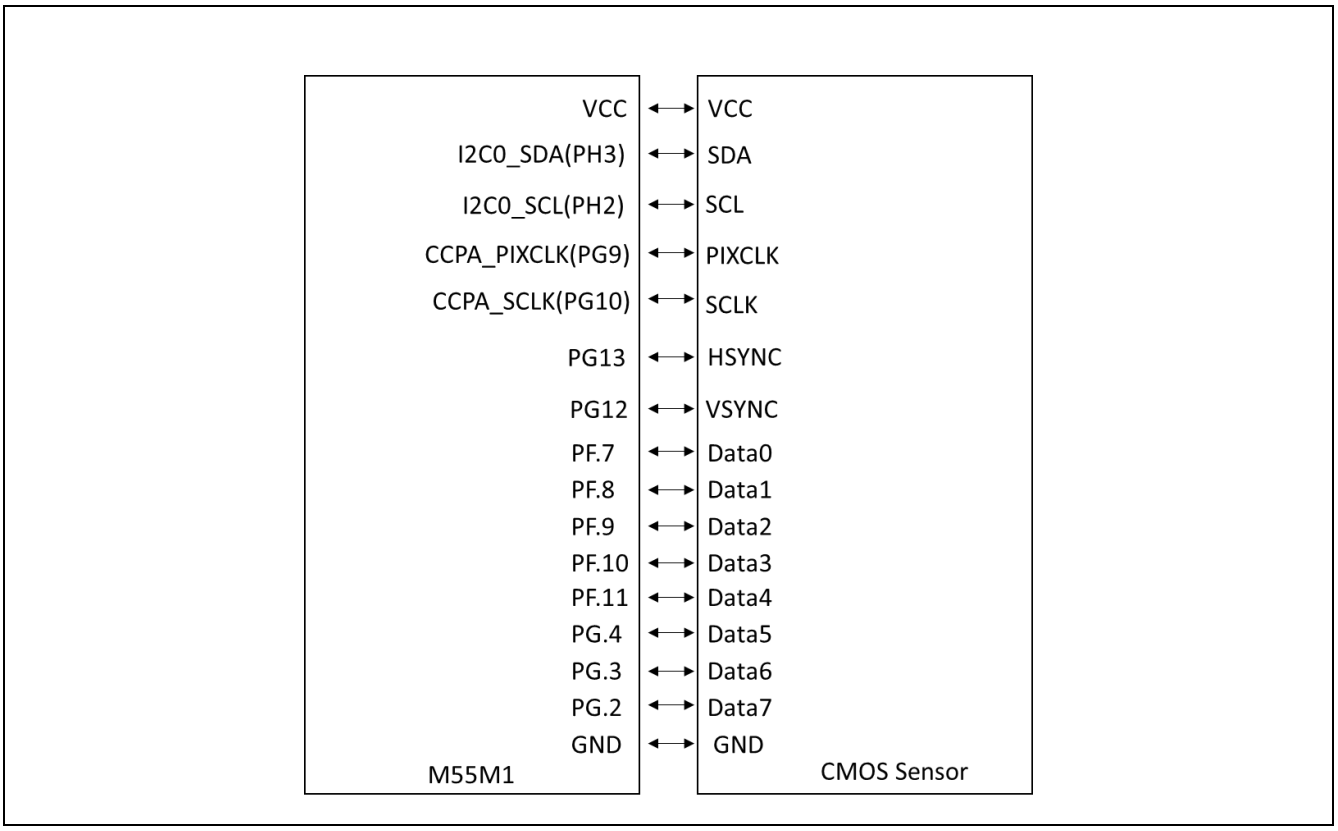

图 3-2 CMOS 线路示意图

4. 目录信息

目录结构如下图所示。

EC_M55M1_Smart_Motion_Detection_V1.00	
Library	Sample code header and source files
CMSIS	Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp.
Device	CMSIS compliant device header file
StdDriver	All peripheral driver header and source files
SampleCode	
MachineLearning	
Project	Source file of example code

图 4-1 目录信息

5. 范例程式执行

1. 根据目录信息章节进入 ExampleCode 路径中的 KEIL 文件夹，双击 *M55M1_Smart_Motion_Detection.uvprojx*。
2. 进入编译模式接口
   - 编译
   - 下载代码至内存
   - 进入 / 离开除错模式
3. 进入除错模式接口
   - 执行代码

6. 修订纪录

Date	Revision	Description
2024.06.24	1.00	初始发布。

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.